Various Code Clone Detection Techniques and Tools: A Comprehensive Survey

Pratiksha Gautam(✉) and Hemraj Saini

Department of Computer Science and Engineering,
Jaypee University of Information Technology, Waknaghat, Solan 173234, India
pratikshamtech20@gmail.com, hemraj.saini@juit.ac.in

Abstract. In this paper, we have discussed several code replication detection methods and tools in different dimensions. This review has provided an extensive survey codec clone detection techniques and tools. Starting from clone perceptions, classification of clones and an overall assortment of selected techniques and tools is discussed. This paper covers the whole paradigm in clone detection and presents open research avenues in code clone detection.

Keywords: Software security · Code clone · Program dependency graph · Detection techniques

1 Introduction

Code segments usually occurs due to replication from one place and then rewrite them in to another section of code with or without variations/changes are software cloning and the copied code is called clone. Various researchers [1–5] have reported more than 20–59% code replication. The problem with such copied code is that an error detected in the original must be checked in every copy for the same bug. Moreover, the copied code expansions the effort to be done when augmenting the code [5, 8]. However, the code quality analysis (improved quality code), replication identification, virus recognition, facet mining, and bug exposure are the other software engineering tasks which require the mining of semantically or syntactically identical code segment to facilitate clone detection significant for software analysis [6]. Fortunately, there are a number of comparison and evaluation studies which are related to numerous clone detection techniques. Recently, Rattan et al. [7], has presented a methodical survey on clone detection while Roy et al. [8] has presented an qualitative comparison and evaluation of clone detection tools and techniques. Bellon et al. [9] has presented an extensive quantitative assessment of six clone detectors which is based on large C and Java programs for clone detection. Further, the potential studies have evaluated the clone detection approach in other context [10–15].

In this paper, we have provided a comprehensive review on presently accessible clone detection approaches and tools. We will start with the basic introduction of code clones after that classify and compare the techniques and tools in two different ways. Foremost, the classification of clone types and their techniques and subsequent categorization of clone detection tools. The remaining of the paper is structured as follows.

A. Unal et al. (Eds.): SmartCom 2016, CCIS 628, pp. 655–667, 2016.
DOI: 10.1007/978-981-10-3433-6_79

The Sect. 2 presents the taxonomy of code clones. The Sect. 3 related to various clone detection methods. The Sect. 4 explores the code clone detection tools. Research gaps are discussed in Sect. 5 and finally, Sect. 6 concludes the paper.

2 Classification of Code Clones

Figure 1 characterizes the taxonomy of code clones. It can be categorized on the basis of three aspects which are illustrated below. Clone classifications are used for expansion reengineering and detection methods. On the basis of clone classification, we have reiterated on the most prominent types of clone, which eventuates at the time of reengineering. In the following, code clones are assorted on basis of three facets such as: (1) similarities between two code segments, (2) clone instance position in program, and (3) refactoring opportunities with the replicated code.

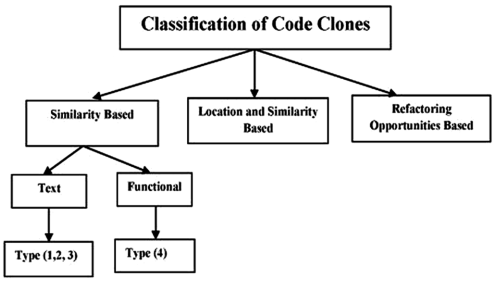
Classification of Code Clones
Similarity Based
Location and Similarity Based
Refactoring Opportunities Based
Text
Functional
Type (1,2, 3)
Type (4)

Fig. 1. Classification of code clones.

The similarity based clones are mainly of two types such as: (1) two code segment can be identical on the basis of similarity of their program content and (2) it can be similar in their functionalities without being textually identical. However, textual similarity based clones are of three types as type-1 (similar code segment without modification except for modification in whitespace and comments) type-2 (structurally/syntactically similar copied code, except for changes in names of function identifiers, variables, types), and type-3 (identical code fragment with or without further modifications; statements were changed, added or removed). The syntactic elements are to be measured in this taxonomy which has been altered by the programmer after replication. For instance, the methods which are same except the name or the methods which are identical for the types of parameters integrated in high-similarity code clones. The type-4 (similar computation but different structure) clone based on similar functionalities.

The similarity between two functions is of three types which are based on four points of comparisons such as name of the function, layout of the code, lexis in the functions and control flow of the functions has been given by Mayrand et al. [3]. A taxonomy for clone methods proposed by Balazinska et al. [16] with 18 different categories which considers each group of clone methods on the basis of differences existing between them. The categories specify the amount of contents of the method has been copied and also what type of syntax elements have been altered. At the first level, two categories based on general similarities such as identical and external changes.

The second instances, the token variations and method aspects based on three categories. The third point based on the significance of the particular token in method body and, moreover the fourth phase is based on token-sequence distinction in function body. The three distinct types of clones such as exact clones, parameterized clones, and clones which have other pervasive features illustrated by Bellon and Koschke [9, 17, 18] for accomplish a good assessment between different detection tools. The objective of this categorization is to analysis the detection and classification adequacy of different clone detection tools. A clone topology with one supplementary type (type-4) presented by Davey et al. [19] which is based on the Bellon and Koschke [9, 17, 18]. In addition to this, the authors [18] detected type-1 to type-3 clones and the detection of type-4 leaving it as future work. Kontogiannis [20] details four types of clones, which are based on functional scheme of replication such as exact clones, the clones which are similar except for analytically replaced with variable names as well as data types; the third is clones with further adaptations. The fourth is clones with statements have been added or deleted.

Further, classification is related to location and similarity of code clones. This categorization is based on place distinctions as well as physical expanse between clone instance positions. The refactoring opportunities or impediments based on the fact that the code segments which are located in the same file, same function or in files from different directories can be improved without affecting their external behavior by using refactoring. The clone instances in object oriented system are to be found at specific position in the class hierarchy. The rudimentary parsing technology is sufficient for extracting such type of assortments for a clone pair. The illustrations of such types of classifications provided by various authors are as follows. A hierarchical categorization of software clones which are consists of three partitions using two aspects such as locations and functionality given by Kapser and Godfrey [21]. The first classification of clones is based on their substantial position in the program text. Second, clones taxonomy is based on the type of realm in which they are located and third is function to function code clones. Monden et al. [22], has described how to simplify the relation between software quality and code by using module-based categorization of code clones and they have also provided taxonomy of modules.

Finally, the refactoring opportunities based classifications discussed the simplicity to extort the copied code from the refactoring perspective. The classifications of such kinds of differences are based on methods which have been defined outside of the copied code fragment and the uses of variables. The context analysis has been proposed by Balazinska et al. [16] to complete the difference analysis of code clones for computer-assisted refactoring. Fanta and Rajlich [23] proposed an approach of reengineering scenarios to eliminate clones, which is based on automated restructuring tools. The object-oriented systems (in SMALLTALK) investigated by Golomingi [24] and the author have also provided a clone relationship scenarios taxonomy based on the class hierarchy relationships of the methods which consists of duplicate code fragments. Basically, there are four types of clone and each type is classified as shown in Table 1 below.

Table 1. Summary of clone taxonomy.

Type 1	Type 2	Type 3	Type 4
Exact clone	Renamed clone	Near-miss clone	Structural clone
Structural clone	Parameterized clone	Gapped clone	Function clone
Function clone	Near-miss clone	Non-contiguous clone	Reordered clone
	Function clone	Reordered clone	Intertwined clone
		Structural clone	Semantic clone
		Function clone	

Table 1 characterizes four types of clones as well as their sub-types. The similarity based taxonomy such as text based (type-1 to 3) and function based (type-4). Type 1 clone has been categories as follows. (1) The exact clone (similar code except some variations in comments), Type 1, 3, 4 as (2) the structural (it is based on level of similarity), Type 1,2,3,4 as (3) function (subset of structural). Type 2 as (1) renamed (modification in copied code), (2) parameterized (renamed clone with renaming). Type 2, 3 as (3) near-miss (slight modifications in copied fragment but syntactic structure remains same). Type 3 as (1) gapped (add, delete, modify some portion between segment), (2) non-contiguous (Like near-miss, and gaps are allowed between code fragment). Type 3, 4 as (3) reordered (some statement have been reordered). Type 4 as (1) intertwined (making two segments in to one segment).

3 Code Clone-Detection Techniques

The clone detection techniques can be analyzed on the basis of code clone properties. There are variant code clone properties which are shown in Fig. 2.

Figure 2 depicts some clone properties such as normalization means apply a number of refinements as remove white space, comments etc. before actual comparison. The source representation, it is the result after the transformation. In the comparison phase the granularities are used for a particular technique. Comparison algorithms play a vital role in detection of dissimilar types of code clones. The complexity based on the types of comparison algorithms as well as types of transformations. The clone similarity means different kinds of clones can be identified by different techniques. The granularity can be fixed or free. The language independency property

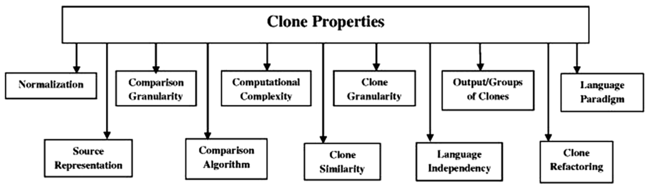

Fig. 2. Various types of code clone properties.

verified language sustain of a detection tool. The output aspect indicates what kind of output will be occurred as clone pair or clone classes or both. Clone refactoring indicates restructuring existing code without changing its external behavior. The language paradigm implies programming paradigm which is targeted for the particular method of interest.

3.1 Classification of Code Clone Detection Techniques

There are following types of clone detection techniques.

(1) The text/string based approach: In this method, the source fragments are analyzed as subsequence of text. The two segments are compared textually with each other on the basis of different transformations like white space, newline and removing comments etc. to locate sequences of same strings. Several researchers have proposed numerous string/text based techniques for clone detection. The lexer as well as line-based string matching algorithm on tokens for the line of text used by Baker [27, 28] with the help of a tool named as Dup. It also used special parameter for identifying clones (which have different variables names). Although, it was not able to detect clone written in different style and it could not support exploration and navigation between the copied codes. Koshke et al. [18] has overwhelmed this problem by using tokens and non-parameterized suffixes. Although, authors were unable to detect the exact and parameterized clones as well as they could not make a distinction between them. Moreover, the clones (text based tool) which is proposed by Koshke et al. [18] does not check whether the identifiers had been renamed consistently. Karp-Rabin fingerprinting algorithm used by Johnson [25] for clone detection as well as to measures the fingerprints of a text for all length substring of the source code. The whole text is partitioned in to a set of substring because of each character in this technique is consists of at least one substring and then raw transformation [30] is applied for matching of those substrings. However, the limitation of this technique is that to keep 50 lines match resulting in to diminish large number of false positive. The island grammar technique used by Cordy et al. [29] for identifying syntactic constructs. Moreover, the author also provided the detection of near miss clones for HTML web pages. The constraints of this technique is that it was unable to normalize any code and it used smallest comparison. The string-based dynamic pattern matching algorithm which is language independent proposed by Ducasse et al. [2]. Further, this technique could not identify meaningful clone resolution in language-independent manner due to the cohesiveness of the code. The latent semantic indexing [31] based approach proposed by Marcuss [26]. This approach detect the clones by extremity its comparison domain within comments and the identifier in spite of compare whole source code. It cannot detect such types of clones which have same structure nevertheless the identifiers name is different. All of the detection approach which have been discussed above shows that it does not apply transformation on the source code, the recent approach which has been proposed by Ducasse et al. [2] has used several transformation on the raw source

code. Although the cost of text based approach is awfully less except the code having identifier changes, line split, amputation of parenthesis, type, etc. cannot be analyzed and identified whether it is a cloned code or not.

(2) The lexical/token-based approach: The token based approaches are also called lexical approach. In this technique, the whole source code is divided in to tokens by lexical analysis and then all the tokens are formed in to a set of token sequence. Finally, the sequence is scanned for identifying duplicated code. One of the foremost tool of token based approach named as CCFinder proposed by Kamiya et al. [32]. Foremost, the lexer partitions each line of text in to tokens and subsequently forms a single token sequence and moreover, the suffix tree matching algorithm is used to find similar sub-sequences of token sequence. Although, Dup is also a token-based approach tool in the sense that it is also used lexer for tokenization as well as for comparisons based on suffix tree matching algorithm proposed by Baker [27, 28]. CP-Miner [34, 35] has been introduced to overwhelm the problem of CCFinder and Dup, in which a frequent subsequence mining technique is used for clone detection rather than sequential analysis in CCFinder and Dup. A plug-in in visual studio based approach which detects clones in Java and C# and it was not able to handle defects from programmer side itself proposed by Juergens et al. [36]. However, the same approach for C ++ and C# was proposed by Kawaguchi et al. [37] and it could not overcome the problem as in [36].

(3) The syntactic/tree-based approach: In this approach, the program is represented in the form of abstract syntax tree (AST) rather than creating tokens for each statement and with the help of tree matching algorithm similar sub-tree is searched in the same tree. One of the initial AST-based tool named as CloneDR proposed by Baxter et al. [38]. It creates AST with the help of compiler generator and then compares its sub tree by using metrics which is based on hash functions. Although, it was not able to detect identical clones. To overcome this problem, the Bauhaus has provided a ccdiml [39] tool by avoiding the uses of hashing and similarity metrics. However, it was incompetent to verify the renamed identifiers. Yang [40] has presented one of the grammar based approach. It is used for finding the syntactic variations between the two versions of the same program by creating their parse tree and then apply dynamic programming technique for identifying similar sub tree. Wahler et al. [41] explored the approach to detect the exact and parameterized clone. This approach foremost convert the AST into XML and subsequently used frequent item set data mining technique [33] for extracting the clones. Evas and Fraser [42] provided a further abstraction of this approach by finding near miss clones as well as exact clones by using only AST leaves rather than whole AST. Even though, it could not detect much of the exact clones. A tool named Clone Tracker in Java was developed by Duala Ekoko et al. [43]. However, it was unable to identify post programming due to the numbers of false positives. A clone management tool in Java which has amplified the time for clone detection proposed by Nguyen [44]. However, aforementioned researches shows that gapped clones could not be find by the AST as well as it could not detect clones if the statements are reordered and does not follow the data flow. The limitations of AST can be easily overcome by the use of PDG-based technique.

(4) The semantic/PDG-based approach: The AST based problems was overwhelmed by the program dependency graph (PDG) [45–47]. The PDG approach used the data flow and control flow [50] for clone detection semantically and syntactically. PDG-DUP is one of the most prominent PDG-based clone detection approach proposed by Komondoor and Horwitz [45, 48]. It is based on program slicing technique for identifying PDG sub graph without changing its semantics behavior. Further, the same slicing based clone analysis approach accomplished by Gallagher and Lucas [49]. They compute program slices on all the variables of a code but could not find any analysis outcome. An iterative approach within PDG proposed by Krinke [46] for identifying maximal similar sub graph but it cannot be used on any type of system to find the clone. According to the several researchers who are using the PDG have concluded that PDG-based techniques can find non-contiguous clones but it cannot be applied to large systems and it will require more time for code clone detection.

(5) The syntactic/metric-based approach: In metric-based methods dissimilar metrics are assembled such as number of functions, number of lines etc. from code segments and then evaluates that metrics in spite of assessments of source code directly. Mayrand et al. [3] computed metrics from expression, layouts and control flow for each function elements of a program and then similar metrics returned as software clones. However, some metrics are not identified in that case they used intermediate representation language (IRL) for exemplifying each function of code. It detects function-based copy-paste instead of segment-based copy-paste which occurs recurrently. The feasible matches identified by an abstract pattern tool which is based on markov model provided by Kontogiannis et al. [4]. The authors used metrics for clone detection which is extorted from an AST of the code and then match detection is done by using dynamic programming. However, it was unable to identify copy-pasted code rather than it only measures similarity between the codes. Further, the identical approach used by Di Lucca et al. [51] for acquisition the similarity between the static HTML pages by evaluating their level of similarity, which is performed by computing the Levenshtein distance of the code [52]. eMetrics tool is used for detecting function clones and then detected clones are clustered according to refactoring opportunities by Calefato Lanuible [53, 54]. Moreover, the mined code is checked manually for finding that is a true positive or not yet it could not be executed on vast systems. So, the authors concluded that metric-based approach can identify simply clones from the code.

(6) The semantic/hybrid approach: There are numerous hybrid methods for code clone detection. The hybrid approach is a collection of several approaches and it can be classified on the basis of preceding techniques. The tree and token based-hybrid approach proposed by Koschke et al. [18] for finding type-I (exact) and type-II code clones. In this approach authors generate suffix tree for serialized AST nodes which is sequentialzed in preorder traversal and then by using suffix tree based algorithm comparisons is performed on the tokens of the AST nodes in place of AST nodes. The Microsoft's new phoenix framework [55], was also used for the detection of function level clones with the same approach. It can detect exact function clone as well as parameterized clone with identifier renaming not data type changes. The analogous approach proposed by Greenan [56] with the

sequence matching algorithm for the detection of method level clones. Jiang et al. [57] explored AST in Euclidean space for computing the vectors as well as group these vectors on the basis of similarity through the Locating Sesitive Hasing (LSH) [58]. A dynamic pattern matching as well as characterization based hybrid approach is provided by Balazinska [59] in which method of each bodies are computed with quality metrics and then evaluated identified clusters by using Patenaude's [60] metric-based approach. A dynamic change tracking and resolution in Java language based novel approach explored by DeWit [61]. Further, it was unable for data flow detection as well as data flows although, it detects the clones at the programmer's level. In addition to this, several other approaches for the clone detection in other context have been proposed in [10–15]. All these approaches mainly emphasized on the detection of type-1, type-2, and type-3 clones. However, aforementioned comprehensive survey has been presented graphically in Fig. 3.

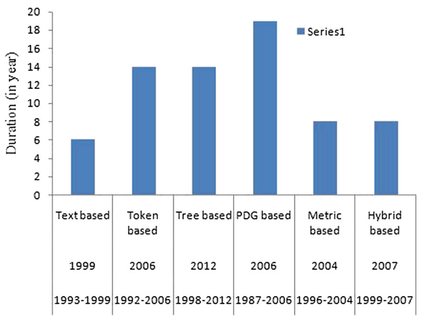

Fig. 3. Comprehensive survey of code clone detection techniques.

4 Comparisons of Clone Detection Tools

The software clone detection tools are multivariate and their abstraction entails a methodical scheme for recounting their property. In this section, we indexed the different clone detection tools presented in the literature in a tabular form as shown in Table 2. The Table 2 illustrates the assessments of various tools and techniques. In this table, the first column represents the author name, 2nd column refers to the tools name,

Table 2. Comparisions of Clone detection tools

Author	Tools	Techniques	Supported-language	Domain
Baker [1, 26]	Dup	Line/Text based	C, C++, Java	CD/Linux
Kamiya, et al. [24]	CCFinder	Transformation/Token comp. with suffix tree	C, C++, Java, COBOL etc.	CD/Windows/NT
Bellon [8]	Ccdiml (Bauhaus)	AST/Tree Matching	C, C++	CD/Linux
Krinke et al. [38]	Duplix	PDG, graph Matching	C?	CD

the 3rd column signifies the proposed technique, 4th column imply whether the tool supported languages and 5th column shows the domain of the tools.

5 Open Research Issues in Code Clone Detection

There is no code clone detection technique for the detection of non-trivial code clone, which is also ideal in terms of portability, scalability, precision and recall. Every tool has its own limitations, making it difficult to define which is realistic for clone detection. The type-1 as well as type-2 clone can be easily detected in comparison of type-3 and type-4. The PDG based-approach can only detect type-3 and type-4 clone but the shortcoming of this algorithm is that it produces many variants of the same clone, thereby taking longer time to process a program. Thus, it is essential for such type of technique and tool that may overcome the limitations of existing techniques for clone detection.

6 Conclusion

The code-clone detection is an emerging issue in the software ecosystem which degrades the software's comprehensibility as well as maintainability. Therefore, its analysis and detection is necessary for improving the quality, structure and design of the software system. In this paper, discussion in terms of attributes based clone categorization, classification of clone detection tools as well as approaches such as text based, token based, tree based, PDG based, metric based and hybrid technique on the basis of their property and sub-property. However, numerous algorithms have been developed based on aforementioned approaches, but sill now the detection of clone with accuracy and efficiency is a potential issue. There are various algorithms for clone detection in which some algorithms are less efficient when a large system is to be compared as well as some algorithms detects only a particular type of clone. This paper presents an extensive comparison of tools and techniques as well as research gaps in clone detection so that one can easily select an appropriate method according to the requirement and can analyze opportunities for hybridizing various techniques that may overcome the existing research gaps in clone detection algorithms.

Acknowledgments. The authors would like to thank Professor Ghanshyam Singh as well as anonymous reviewers for critical comments and suggestions to improve the quality of the manuscript.

References

1. Baker, B.S.: On finding duplication and near-duplication in large software systems. In: Proceedings of 2nd IEEE Working Conference on Reverse Engineering, Toronto, Ontario, Canada, pp. 86–95, July 1995
2. Ducasse, S., Rieger, M., Demeyer, S.: A Language independent approach for detecting duplicated code. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Oxford, UK, ICSM 1999, pp. 109–118 (1999)
3. Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Monterey, CA, pp. 244–254 (1996)
4. Kontogiannis, K., Mori, R.D., Merlo, E., Galler, M., Bernstein, M.: Pattern matching for clone and concept detection. J. Autom. Softw. Eng. **3**(1), 79–108 (1996)
5. Lague, B., Proulx, D., Mayrand, J., Merlo, E.J., Hudepohl, J.: Assessing the benefits of incorporating function clone detection in a development process. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Washington, DC, USA, pp. 314–321 (1997)
6. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Technical report 541, Queen's University at Kingston (2007)
7. Rattan, D., Bhatia, R., Singh, M.: Software Clone detection: a systematic review. Inf. Softw. Technol. **55**(7), 1165–1199 (2013)
8. Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. Sci. Comput. Program. **74**(7), 470–495 (2009)
9. Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.: Comparison and evaluation of clone detection tools. IEEE Trans. Softw. Eng. **33**(9), 577–591 (2007)
10. Patil, R.V., Joshi, S., Shinde, S.V., Ajagekar, D.A., Bankar, S.D.: Code clone detection using decentralized architecture and code reduction. In: Proceedings of IEEE International Conference on Pervasive Computing, Pune, India (ICPC 2015), pp. 1–6, January 2015
11. Keivanloo, I., Zhang, F., Zou, Y.: Threshold-free code clone detection for a large-scale heterogeneous Java repository. In: Proceedings of 22nd IEEE International Conference on Software Analysis, Evolution and Reengineering, Montreal, QC (SANER 2015), pp. 201–210 (2015)
12. Chodarev, S., Pietrikova, E., Kollar, J.: Haskell clone detection using pattern comparing algorithm. In: Proceedings of 13th IEEE International Conference on Engineering of Modern Electric Systems (EMES 2015), Oradea, Romania, pp. 1–4 (2015)
13. Kamiya, T.: An execution-semantic and content-and-context-based code-clone detection and analysis. In: Proceedings of 9th International Workshop on Software Clones, Montreal, QC (IWSC 2015), pp. 1–7 (2015)
14. Singh, M., Sharma, V.: Detection of file level clone for high level cloning. In: Proceedings of 3rd Elsevier International Conference on Recent Trends in Computing (ICRTC 2015), India, pp. 915–922 (2015)
15. Basit, H.A., Jarzabek, S.: A data mining approach for detecting higher-level clones in software. IEEE Trans. Softw. Eng. **35**(4), 497–514 (2009)
16. Balazinska, M., Merlo, E., Dagenais, M., Lague, B., Kontogiannis, K.: Measuring clone based reengineering opportunities. In: Proceedings of the 6th IEEE International Symposium on Software Metrics (METRICS 1999), USA, pp. 292–303, November 1999
17. Bellon, S:. Vergleich von techniken zur erkennung duplizierten quellcodes. Master's thesis no. 1998, University of f Stuttgart (Germany). Institute for Software Technology, September 2002

18. Koschke, R., Falke, R., Frenzel, P.: Clone detection using abstract syntax suffix trees. In: Proceedings of the 13th IEEE Working Conference on Reverse Engineering, Italy, pp. 253–262, October 2006
19. Davey, N., Barson, P., Field, S., Frank, R., Tansley, D.: The development of a software clone detector. J. Appl. Softw. Technol. **1**(3/4), 219–236 (1995)
20. Kontogiannis, K.: Evaluation experiments on the detection of programming patterns using software metrics. In: Proceedings of the 4th IEEE Working Conference on Reverse Engineering, Netherlands, pp. 44–54, October 1997
21. Kapser, C., Godfrey, M.W.: Aiding comprehension of cloning through categorization. In: Proceedings of the 7th IEEE International Workshop on Principles of Software Evolution, Japan, pp. 85–94, September 2004
22. Monden, A., Nakae, D., Kamiya, T., Sato, S.I., Matsumoto, K.I.: Software quality analysis by code clones in industrial legacy software. In: Proceedings of 8th IEEE International Symposium on Software Metrics, Canada, pp. 87–94, June 2002
23. Fanta, R., Rajlich, V.: Removing clones from the code. J. Softw. Maintenance **11**(4), 223–243 (1999)
24. Koni-N'sapu, G.G.: A scenario based approach for refactoring duplicated code in object oriented systems. Diploma thesis, University of Bern, Germany (2001)
25. Johnson, J.H.: Identifying redundancy in source code using fingerprints. In: Proceeding of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, Canada, pp. 171–183, October 1993
26. Marcus, A., Maletic, J.: Identification of high-level concept clones in source code. In: Proceedings of 16th IEEE International Conference on Automated Software Engineering (ASE 2001), pp. 107–114, November 2001
27. Baker, B.S.: A program for identifying duplicated code.de. In: Proceedings of Computing Science and Statistics, 24th Symposium on the Interface, pp. 49–57, March 1993
28. Baker, B.S.: Parameterized difference. In: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), Maryland, USA, pp. 854–855, January 1999
29. Cordy, J.R., Dean, T.R., Synytskyy, N.: Practical language-independent detection of near-miss. In: Proceedings of the 14th Conference of the Centre for Advanced Studies, Canada, pp. 1–12, October 2004
30. Cox, I.J., Linnartz, J.P.M.: Some general methods for tampering with watermarks. J. Sel. Area Commun. **16**(4), 587–593 (1998)
31. Dumais, S.T.: Latent Semantic Indexing (LSI) and TREC-2. In: Proceedings of the 2nd Text Retrieval Conference (TREC 1994), Maryland, pp. 105–115, March 1994
32. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Softw. Eng. **28**(7), 54–67 (2002)
33. Baker, B.S.: Finding clones with dup: analysis of an experiment. IEEE Trans. Softw. Eng. **33** (9), 608–621 (2007)
34. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-miner: a tool for finding copy-paste and related bugs in operating system code. In: Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), Berkeley, CA, USA, vol. 4, no. 19, pp. 289–302, December 2004
35. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans. Softw. Eng. **32**(3), 176–192 (2006)
36. Juergens, E., Deissenboeck, F., Hummel, B.: Clone detective - a workbench for clone detection research. In: Proceedings of the 31st IEEE International Conference on Software Engineering, Vancouver, BC, pp. 603–606 (2009)

37. Kawaguchi, S., Yamashina, T., Uwano, H., Fushida, K., Kamei, Y., Nagura, M., Iida, H.: SHINOBI: a tool for automatic code clone detection in the idea. In: Proceedings of 16th IEEE Working Conference on Reverse Engineering (WCRE 2009), Lille, pp. 313–314 (2009)
38. Baxter, I.D., Yahin, A., Moura, L, Anna, M.S.: Clone detection using abstract syntax trees. In: Proceedings of the 14th IEEE International Conference on Software Maintenance (ICSM 1998), Maryland, pp. 368–377, November 1998
39. Raza, A., Vogel, G., Plödereder, E.: Bauhaus – a tool suite for program analysis and reverse engineering. In: Pinho, L.M., González Harbour, M. (eds.) Ada-Europe 2006. LNCS, vol. 4006, pp. 71–82. Springer, Heidelberg (2006). doi:10.1007/11767077_6
40. Yang, W.: Identifying syntactic differences between two programs. J. Softw. Prac. Exp. **21** (7), 739–775 (1991)
41. Wahler, V., Seipel, D., Fischer, G.: Clone detection in source code by frequent item set techniques. In: Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2004), USA, pp. 128–135, September 2004
42. Evans, W.S., Fraser, C.W., Ma, F.: Clone detection via structural abstraction. J. Softw. Qual. **17**(4), 309–330 (2009)
43. Duala-Ekoko, E., Robillard, M.P.: Clone tracker: tool support for code clone management. In: Proceedings of the 30th ACM International Conference on Software Engineering, Washington, DC, USA, pp. 843–846 (2008)
44. Nguyen, H.A., et al.: Clone management for evolving software. IEEE Trans. Softw. Eng. **38** (5), 1008–1026 (2012)
45. Komondoor, R., Horwitz, S.: Using slicing to identify duplication in source code. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 40–56. Springer, Heidelberg (2001). doi:10.1007/3-540-47764-0_3
46. Krinke, J.: Identifying similar code with program dependence graphs. In: Proceedings of the 8th IEEE Working Conference on Reverse Engineering (WCRE 2001), Germany, pp. 301–309, October 2001
47. Liu, C., Chen, C., Han, J., Yu, P.S.: GPLAG: detection of software plagiarism by program dependence graph analysis. In: Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2006), Philadelphia, pp. 872–881, August 2006
48. Komondoor, R.V.: Automated duplicated-code detection and procedure extraction. Doctoral thesis, University of Wisconsin- Madison, USA (2003)
49. Gallagher, K., Layman, L.: Are decomposition slices clones? In: Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC 2003), USA, pp. 251–256, May 2003
50. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. **9**(3), 319–349 (1987)
51. Di Lucca, G.A., Di Penta, M., Fasolino, A.R., Granato, P.: Clone analysis in the web era: an approach to identify cloned web pages. In: Proceedings of the 7th IEEE Workshop on Empirical Studies of Software Maintenance, Italy, pp. 107–113, November 2001
52. Di Lucca, G.A., Di Penta, M., Fasolino, A.R., Granato, P.: An approach to identify duplicated web pages. In: Proceedings of the 26th International Conference on Computer Software and Applications, England, pp. 481–486, August 2002
53. Calefato, F., Lanubile, F., Mallardo, T.: Function clone detection in web applications: a semi automated approach. J. Web Eng. **3**(1), 3–21 (2004)
54. Lanubile, F., Mallardo, T.: Finding function clones in web applications 2003. In: Proceedings of 7th IEEE European Conference on Software Maintenance and Reengineering (CSMR 2003), Italy, pp. 379–386, March 2003

55. Tairas, R., Gray, J.: Phoenix-based clone detection using suffix trees. In: Proceedings of the 44th ACM Annual Southeast Regional Conference (ACM-SE 2006), Melbourne, pp. 679–684, March 2006
56. Greenan, K.: Method-level code clone detection on transformed abstract syntax trees using sequence matching algorithms. Student report, University of California, Santa Cruz, USA (2005)
57. Jiang, L., Misherghi, G., Su, Z., Glondu, S.: Scalable and accurate tree-based detection of code clones. In: Proceedings of the 29th IEEE International Conference on Software Engineering (ICSE 2007), USA, pp. 96–105, May 2007
58. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the 20th ACM Annual Symposium on Computational Geometry (SoGG 2004), New York, pp. 253–262, June 2004
59. Balazinska, M., Merlo, E., Dagenais, M., Lagüe, B., Kontogiannis, K.: Measuring clone based reengineering opportunities. In: Proceedings of the 6th IEEE International Software Metrics Symposium (METRICS 1999), Florida, USA, pp. 292–303, November 1999
60. Patenaude, J.F., Merlo, E., Dagenais, M., Laguë, B.: Extending software quality assessment techniques to java systems. In: Proceedings of the 7th IEEE International Workshop on Program Comprehension (IWPC 1999), USA, pp. 49–56, May 1999
61. De Wit, M., Zaidman, A., Van Deursen, A.: Managing code clones using dynamic change tracking and resolution. In: Proceedings of IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, AB, pp. 169–178 (2009)